Greedy Q-Learning Agent

Consider a greedy (non-exploratory) variant of the Q-Learning agent, deciding by

$$a_{k+1} = \arg\max_a Q^\pi(s_{k+1}, a)$$

Here, the iteration

$$Q^\pi(s_k, a_k) \leftarrow Q^\pi(s_k, a_k) + \alpha \left(r_k + \gamma \max_a Q^\pi(s_{k+1}, a) - Q^\pi(s_k, a_k) \right)$$

can get rid of the maximization:

$$Q^\pi(s_k, a_k) \leftarrow Q^\pi(s_k, a_k) + \alpha \left(r_k + \gamma Q^\pi(s_{k+1}, a_{k+1}) - Q^\pi(s_k, a_k) \right)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

SARSA Agent

SARSA agent is the exporatory Q-Learning agent where even for a non-greedy strategy the iteration is changed to

$$Q^\pi(s_k, a_k) \leftarrow Q^\pi(s_k, a_k) + \alpha\left(r_k + \gamma Q^\pi(s_{k+1}, a_{k+1}) - Q^\pi(s_k, a_k)\right)$$

Name due to *S*tate-*A*ction-*R*eward-*S*tate-*A*ction quintuplet

$$s_k, a_k, r_k, s_{k+1}, a_{k+1}$$

from which Q^π iterated.

Q-Learning is an *off-policy* (as in, less dependent on policy) strategy. Tends to learn Q better event if π is far from optimal.

SARSA is an *on-policy* strategy. Tends to adapt better to partially enforced policies.

Problems with Table Models

So far, $\widehat{U}, \widehat{Q}$ have been look-up tables (arrays) demanding at least $\mathcal{O}(|S|)$ resp. $\mathcal{O}(|S| \cdot |A|)$ memory and time.

Table-based agents would not scale to large ('real-life') state spaces S.

- Backgammon or Chess: $|S|$ somewhere btw. 10^{20} and 10^{45}
- No way to capture in an array, let alone do policy evaluation

A more compact ('generalized') model for

$$U : S \to \mathbb{R} \text{ or } Q : S \times A \to \mathbb{R}$$

is needed. Must allow learning (updating) from $[s_k, a_k, r_k, s_{k+1}]$ or $[s_k, a_k, r_k, s_{k+1}, a_{k+1}]$ samples.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Feature-Based Representation of $\widehat{U}$

Consider learning $\widehat{U}$ with the *Direct Utility Estimation* agent.

A simple option is to define a set of relevant features $\phi^i : S \to \mathbb{R}$ and use a *regression model*.

$$\widehat{U}(\mathbf{w}, s) = \sum_{i=1}^{n} w^i \phi^i(s)$$

and adapt the parameters $\mathbf{w} = [w^1, w^2, \ldots, w^n]$ at each episode's end to reduce the squared error

$$E_j(\mathbf{w}, s) = \frac{1}{2}\left(\widehat{U}(\mathbf{w}, s) - u_j(s)\right)^2$$

where $u_j(s)$ is the utility sample obtained for s at the end of episode $j = 1, 2, \ldots$ (when a terminal state is reached).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Feature-Based Representation of $\widehat{U}$ (cont'd)

Going against the error gradient with learning rate $\alpha \in \mathbb{R}$:

$$w^i \leftarrow w^i - \alpha \frac{\partial E_j(\mathbf{w}, s)}{\partial w^i} = w^i + \alpha \left(u_j(s) - \widehat{U}(\mathbf{w}, s) \right) \frac{\partial \widehat{U}(\mathbf{w}, s)}{\partial w^i}$$

Example: Let $[\phi^1(s), \phi^2(s)] = [s^1, s^2]$, i.e., the agent's coordinates in the grid environment and $\phi^3 \equiv 1$.

Note: superscript component indexes to disambiguate from subscripted time indexes

Then

$$\widehat{U}(\mathbf{w}, s) = w^1 s^1 + w^2 s^2 + w^3$$

and the iterative update:

$$\begin{aligned} w^1 &\leftarrow w^1 + \alpha(u_j(s) - \widehat{U}(\mathbf{w}, s))s^1, \\ w^2 &\leftarrow w^2 + \alpha(u_j(s) - \widehat{U}(\mathbf{w}, s))s^2 \\ w^3 &\leftarrow w^3 + \alpha(u_j(s) - \widehat{U}(\mathbf{w}, s)) \end{aligned}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Feature-Based Representation of $\widehat{U}$: Notes

1. Observe:
$$\frac{\partial \widehat{U}(\mathbf{w}, s)}{\partial w^i} = \phi^i(s)$$
So the derivative is simple even with non-linear features such as
$$\phi^i(s) = \sqrt{(s^1 - 4)^2 + (s^2 - 3)^2}$$
measuring the Euclidean ('air') distance to the terminal state $(4, 3)$.
2. Features allow to deal with a kind of *partial state observability*. If a component of the state is not observable, design features that do not use that component.

Feature-Based Representation of $\widehat{Q}$

A similar strategy can be applied in the TD agent or the Q-Learning agent. For the latter

$$\widehat{Q}(\mathbf{w}, s, a) = \sum_{i=1}^{n} w^i \phi^i(s, a)$$

where ϕ^i are predefined features of state-action pairs.

Follow the gradient descent (again, $\frac{\partial \widehat{Q}(\mathbf{w},s,a)}{\partial w^i} = \phi^i(s, a)$) at each time k

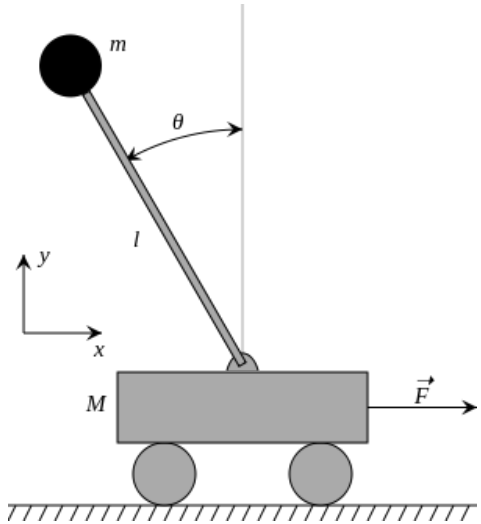
$$w^i_{k+1} = w^i_k + \alpha \left(r(s_k) + \gamma \max_a \widehat{Q}(\mathbf{w}_k, s_{k+1}, a) - \widehat{Q}(\mathbf{w}_k, s_k, a_k) \right) \phi^i(s_k, a_k)$$

The principle is simple, the art is in designing good features ϕ^i.

Inverted Pendulum Demo

Real-valued features especially appropriate where environment is a dynamic physical system. Typical features are *positions* and *accelerations* of objects.

Example: inverted pendulum



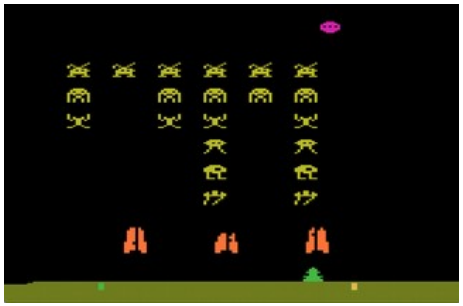
Videos: Single (Experience Replay - see later), Triple (!).

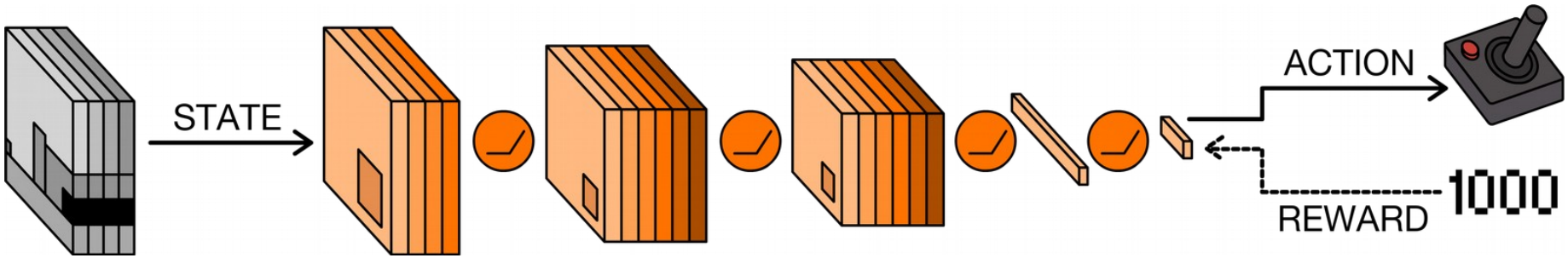
Deep Q-Learning: DQN

Learns to play ATARI 2600 games from screen images and score.
(DeepMind / Nature, 2015)

Deep feed-forward network approximating $Q(s, a)$

- input = state = 4 time-subsequent 84x84 gray-scale screens
- separate output for each $a \in A$
- 2 convolution + 1 connected hidden layers



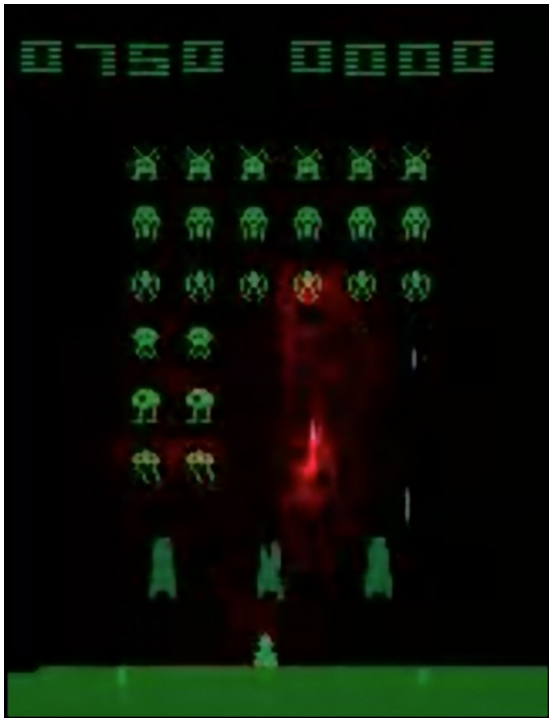


Demo

Deep Q-Learning: DQN (cont'd)

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for
end for

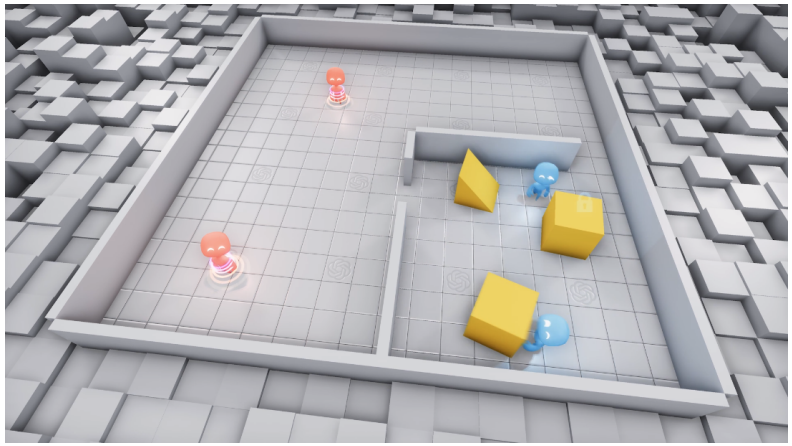


Experience replay prevents long chains of correlated training examples by sampling from a buffer of $\vec{\phi}(s_k), a_k, r_k, \vec{\phi}(s_{k+1})$ tuples recorded in the past.

Backpropagation to the original image inputs reveals areas of 'attention'.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

OpenAI: Hide and Seek Game



(click to visit)

- Two hiders, two seekers, each learning by reinforcement
- Can move, shift and lock blocks, see others and blocks (if in line of sight), sense distance
- Team-wide rewards to hiders: -1 if any hider seen by a seeker, +1 if all hiders hidden
- Seekers get opposite rewards.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Policy Search

Instead of searching $\widehat{Q}$ (or $\widehat{U}$ or P_S and r) search directly a good policy $\pi : S \to A$.

If S unmanageably large, use features again: $\phi^i : S \to Z_i$ where $i = 1, 2, \ldots n$ and Z_i are the feature value ranges.

Then set $\pi(S) = \pi'(\phi_1(S), \ldots \phi_n(S))$ where $\pi' : Z_1 \times \ldots \times Z_n \to A$

Quality of π' is estimated as mean total rewards over repeated episodes using π. Search may e.g. be greedy adjustments to π' improving its quality.

Since A is finite (discrete), π' is not differentiable so gradient descent not applicable. If Z_i are finite (e.g. discretized), this means combinatorial search.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Differentiable Policy Search

Gradient-based policy search is possible with a *stochastic policy* choosing action a in state s with (softmax) probability

$$\frac{e^{q(\mathbf{w},\boldsymbol{\phi}(s,a))}}{\sum_{a' \in A} e^{q(\mathbf{w},\boldsymbol{\phi}(s,a'))}}$$

where $\mathbf{w} \in \mathbb{R}^n$ are real parameters, $\boldsymbol{\phi} = \langle \phi_1, \ldots, \phi_{n'} \rangle$ are some real-valued features, and $q : \mathbb{R}^{n+n'} \to \mathbb{R}$. If q differentiable in all ϕ^i as in e.g. $(n = n')$

$$q(\mathbf{w}, \boldsymbol{\phi}(s,a)) = \sum_{i=1}^{n} w^i \phi^i(s,a)$$

then $\mathbf{w}$ can be adapted through (empirical) gradient descent (but gradient estimation not trivial with stochastic environment and policy).

Reminds of $\widehat{Q}$ learning but q optimized w.r.t. policy performance.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learning a Feature-Based Environment Model

An agent deriving policy from a model of P_S and r can learn such models. In the ADP agent, such models were just relative frequencies.

They can also be feature based. So $P_S(s'|s,a)$ can be modeled e.g. by

$$f(\mathbf{w}, \boldsymbol{\phi}(s', s, a)) = \sum_{i=1}^{n} w^i \phi^i(s', s, a)$$

where ϕ^i are features of the s', s, a triple and w^i real parameters. Gradient method applicable, every transition provides a sample. No need to normalize f (probability!) if only used in arg max expressions.

Similarly for the reward model (modeling a function, not a prob.)

Bayesian Learning of an Environment Model

Consider the following *Bayesian* approach which involves

- a countable probability *distribution class* $\mathcal{M}$ ("model class")
- at each time k, a probability distribution B_k on $\mathcal{M}$ where $B_k(P)$ ($P \in \mathcal{M}$) quantifies the belief that $P_S \equiv P$. B_1 is the initial belief.

At each time k, our model $\xi_k(s_{k+1}|s_k, a)$ of $P_S(s_{k+1}|s_k, a_k)$ is

$$\xi_k(s_{k+1}|s_k, a_k) = \sum_{P \in \mathcal{M}} P(s_{k+1}|s_k, a_k) B_k(P)$$

i.e, a probability-weighted sum where each model contributes the stronger the higher its belief. $|\mathcal{M}|$ may be ∞ but the sum obviously converges.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning of an Environment Model (cont'd)

At each time $k+1$, B_k is updated by the Bayes rule to the posterior

$$B_{k+1}(P) = \alpha P(s_{k+1}|s_k, a_k) B_k(P)$$

for each $P \in \mathcal{M}$, where the normalizer α is such that

$$\sum_{P \in \mathcal{M}} B_{k+1}(P) = 1$$

Note that the s_{k+1} states are sampled mutually independently *given* s_k, a_k from the same distribution $P_S(s_{k+1}|s_k, a_k)$ although s_{k+1} are not independent of s_k or a.

(This can also be posed as learning a separate model $P_{s_k,a_k}(s_{k+1})$ for each possible $s_k \in S, a_k \in A$.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE